Package ‘splusTimeSeries’

April 30, 2021

Version 1.5.2

Title Time Series from S-PLUS

Depends R (>= 2.10), splusTimeDate (>= 2.5.2)

Imports graphics, methods, stats

Description A collection of classes and methods for working with time series.
The code was originally available in S-PLUS.

License BSD_3_clause + file LICENSE

URL https://github.com/spkaluzny/splusTimeSeries

BugReports https://github.com/spkaluzny/splusTimeSeries/issues

LazyData no

NeedsCompilation yes

Author Stephen Kaluzny [aut, cre],
TIBCO Software Inc. [aut, cph]

Maintainer Stephen Kaluzny <spkaluzny@gmail.com>

Repository CRAN

Date/Publication 2021-04-30 10:40:10 UTC

R topics documented:

`aggregateSeries` *Time Series and Signal Aggregation*

Description

Aggregation and coursening of time series and signals. This is the method for the `aggregate` function for objects of class `timeSeries` and `signalSeries`.

Usage

```
aggregateSeries(x, pos, FUN, moving=FALSE, together=FALSE, drop.empty=TRUE,
                include.ends=FALSE, adj, offset, colnames, by, k.by=1,
                week.align=NULL, holidays=timeDate(), align.by=TRUE,
                incr=1, ...)
```

Arguments

`x`	the series object to be aggregated.
`pos`	the break positions for aggregation (can also be supplied in the by argument if desired).
`FUN`	the function to use for aggregation. Often `mean`, `sum`, or `hloc`. If `moving` is FALSE, `FUN` can also be a character string like `"fastFFF"`, to aggregate column-wise using the corresponding fast `igroupFFF` function. (The `together` argument is ignored.) Possible choices for `FFF` are currently `Means`, `Maxs`, `Mins`, `Sums`, `Prods`, `Anys`, and `Alls`.
`moving`	either `FALSE` to do standard aggregation, or a positive integer `N` to perform a moving aggregation (normally used for a moving average) over `N` samples.
`together`	a logical value. if `TRUE`, passes all columns of `x` together into `FUN`; If `FALSE` (the default), passes each column separately into `FUN` for each aggregation block.

`drop.empty`	a logical value. If `TRUE` (the default), drops aggregation blocks with no positions to aggregate from the output.
`include.ends`	a logical value. If `TRUE`, includes the positions before the first given aggregation block and after the last in the first/last blocks. If `FALSE` (the default), does not include those positions in the output.
`adj`	if provided, adjusts the positions of the output series so that they lie a fraction `adj` towards the blocks ending position. The default is to use the lower end of the block for the output position. 0.5 uses the center of the aggregation block for the output position, and 1.0 uses the upper end of the block.
`offset`	as an alternative to `adj`, provide a constant offset to add to the lower end of the aggregation block to get the output series positions. For instance, in monthly aggregation, you might supply an offset of 9 days so that the output positions fell on the tenth of each month.
`colnames`	new column names for the output series. The default is to use the same names as the input series if the output series has the same width.
`by`	if `pos` is missing and `by` is supplied for aggregating calendar-based time series, new positions are generated as a regular time/date sequence using `by`, `k.by`, `week.align`, and `holidays`.

`by` gives the spacing between successive values in the sequence. This can be a `timeSpan`, `timeRelative`, or `numeric` value, in which case `k.by` is ignored.

Alternatively, it can be one of the following character strings:

- `"milliseconds"`
- `"seconds"`
- `"minutes"`
- `"hours"`
- `"days"`
- `"weekdays"`
- `"bizdays"`
- `"weeks"`
- `"months"`
- `"quarters"`
- `"years"`

giving the time units of intervals between values in the sequence.

`k.by`	a non-zero integer giving the width of the interval between consecutive values in the sequence in terms of the units given in `by`. Ignored if `by` is not a character string or if `pos` is given.
`week.align`	if not `NULL` and `by` is `"weeks"`, you can supply a character string (or a number, 0 to 6 with 0 being Sunday) to specify a weekday to use for aggregation. The character string must be sufficient to make a unique case-insensitive match to the strings in `timeSeriesOptions("time.day.name")`. Ignored if `by` is not a character string or `pos` is given.
`holidays`	holidays for business day sequences. (Ignored if `by` is not a character string or `pos` is given.)

`align.by`	a logical value. If `TRUE` (the default), adjustd the sequence so that each element is on a whole number of the `by * k.by` units. For example, if the units are 2 months, make the sequence be only on the first of January, March, and so on. Ignored if `by` is not a character string.
`incr`	For moving aggregation, the moving window moves forward by this many steps in the time series, for each window.
`...`	Additional arguments to pass to FUN.

Value

returns a new time series whose positions are the adjusted passed-in positions or positions generated from `by`, `k.by`, and so on, (or possibly a subset if `drop.empty` is `TRUE`) and whose rows are aggregated from `x` as specified in the arguments. Aggregation takes place by separating `x` into blocks separated by the positions (or overlapping blocks with a fixed number of samples if `moving` is supplied), and then applying `FUN` to each column (or all columns together if `together` is `TRUE`) and forming a new time series with the positions and aggregated data.

See Also

`timeSeries`, `signalSeries`, `align`, `aggregate`

Examples

```
x <- timeSeries(data.frame(1:20,rnorm(20)), timeCalendar(d=1:20))
aggregate(x, FUN=mean, by="weeks")
```

---

align *Time Series and Signal Interpolation and Alignment*

Description

Aligns or interpolates a time series or signal to new positions.

Usage

```
align(x, pos, how="NA", error.how="NA", localzone=FALSE, matchtol=0, by,
      k.by=1, week.align=NULL, holidays=timeDate())
```

Arguments

`x`	the object to be aligned or interpolated.
`pos`	the new positions to align or interpolate it to (either `pos` or `by` is required).
`how`	specifies how to treat unmatched positions. Must be one of the following:

`"NA"`	Insert a row of `NA`.
`"drop"`	Drop that position entirely.
`"nearest"`	Use the row with the nearest position.

`"before"`	Use the data from the row whose position is just before the unmatched position.
`"after"`	Use the data from the row whose position is just after the unmatched position.
`"interp"`	Interpolate linearly between `"before"` and `"after"`.

`error.how`	specifies available actions when an out of bounds error occurs. (Such an error can occur when `how` is `"before"`, `"after"`, or `"interp"`). Must be one of the following:

`"NA"`	Insert a row of `NA`.
`"drop"`	Drop that position entirely.
`"nearest"`	Use the row with the nearest position.

`localzone`	if T (`x` must be a calendar-based time series), align by matching/interpolating with both `x` and `pos` in their local time zones, instead of with the absolute GMT times.
`matchtol`	the tolerance for matching positions. Positions that match within `matchtol` do not invoke the `how` argument methods.
`by`	if `pos` is missing and `by` is supplied for aligning a calendar-based time series, new positions are generated as a regular time/date sequence using one of the following: • `by` • `k.by` • `week.align` • `holidays` `by` gives the spacing between successive values in the sequence. This can be a `timeSpan`, `timeRelative`, or `numeric` value, in which case `k.by` is ignored. Alternatively, it can be one of the following character strings: • `"milliseconds"` • `"seconds"` • `"minutes"` • `"hours"` • `"days"` • `"weekdays"` • `"bizdays"` • `"weeks"` • `"months"` • `"quarters"` • `"years"` These strings give the time units of intervals between values in the sequence.
`k.by`	a non-zero integer giving the width of the interval between consecutive values in the sequence in terms of the units given in `by`. Ignored if `by` is not a character string or if `pos` is given.

week.align	if not NULL, and by is "weeks", you can supply a character string (or a number, 0 to 6, with 0 being Sunday) to specify a weekday to align to. The character string must be sufficient to make a unique case-insensitive match to the strings in timeSeriesOptions("time.day.name"). Ignored if by is not a character string or if a pos is given.
holidays	the holidays for business day sequences. (Ignored if by is not a character string or if pos is given.)

Details

If either x or pos (or the generated sequence) has zero length, a zero-length series is returned.

Value

returns a new time series or a signal whose positions are the passed-in positions or positions generated from by, k.by, and so on, and whose rows are derived from x as specified in the arguments. (Can be a subset if how or error.how is "drop".)

See Also

timeSeries, signalSeries, positions, seriesMerge.

Examples

```
a <- signalSeries(pos=1:10, data=data.frame(a = 11:20, b = 5 * (1:10)))
align(a, c(.2, 3, 7.8, 12), how = "interp", error.how = "nearest")
a <- timeSeries(pos=as(1:10, "timeDate"),
data=data.frame(a = 11:20, b = 5 * (1:10)))
alpos <- as(c(.2, 3, 7.8, 12), "timeDate")
alpos@time.zone <- "JST"
positions(a)@time.zone <- "PST"
align(a, alpos, matchtol = 1, localzone = TRUE)
align(a, matchtol=1, localzone=TRUE, by="days", k.by=2)
```

as.rectangular *Uniform Rectangular Data Functions*

Description

Functions that allow you to access all rectangular data objects in the same way. Rectangular data objects include matrices, data frames and vectors.

Usage

```
as.rectangular(x)
as.char.rect(x)
is.rectangular(x)
subscript2d(x,i,j)
```

```
subscript2d(x,i,j) <- value
numRows(x)
numRows(x) <- value
numCols(x)
numCols(x) <- value
rowIds(x)
rowIds(x) <- value
colIds(x)
colIds(x) <- value
```

Arguments

`x`	the object to be converted to rectangular data (`as.rectangular`), or a rectangular data object.
`i`	the first (row) subscript.
`j`	the second (column) subscript.
`value`	the object to be assign to `x`

Details

`subscript2d`, `numRows`, `numCols`, `rowIds`, `colIds` can also be used on the left side of assignments. The `value` can be a character vector, or anything that can be coerced to a character vector.

- `subscript2d` is for subscripting. When `subscript2d` is used in an assignment, it does not allow subscript replacement outside the bounds of `x`. Instead, set `numRows` or `numCols` first.
- When `numRows` or `numCols` is used in an assignment, the row and column IDs are maintained to have the correct length. Usually, this is done by setting `numRows` on the ID vector, but for some objects (for example, data frames) this might not be appropriate, and they have their own methods.
- Functions `colnames<-` and `rownames<-` simply call `colIds<-` and `rowIds<-`, respectively.
- `as.rectangular` converts any object to a rectangular data object (usually a data frame), if possible.
- `is.rectangular` tests whether an object is rectangular.
- `numRows` and `numCols` count the number of rows and columns.
- `rowIds` and `colIds` (and `rownames` and `colnames`) return the row and column names or other identifiers attached to rows and columns.
- `colnames` and `rownames` return the same values as `colIds` and `rowIds`, respectively, if `do.NULL=T`.
- Instead of using `names` to replace row names from a matrix, use `rowIds` or `dimnames`.
- The functions `colnames`, `rownames`, `colnames<-`, `rownames<-` emulate **R** functions of the same names.

Value

`as.rectangular`	returns `x` if it is already rectangular, or `as.data.frame(x)` if it is not.

`as.char.rect`	takes a rectangular object and returns a rectangular object (vector or matrix) consisting of character strings, suitable for printing (but not formatted to fixed width).
`is.rectangular`	returns `TRUE` if `x` is rectangular, and `FALSE` if it is not.
`subscript2d(x,i,j)`	is like `x[i,j,drop=F]`, except that it allows `x[,1]` (for example) for atomic vectors. Usually, it returns an object of the same class as `x` (this is not appropriate for some objects, such as `"bs"` objects). It does not support a `drop` argument.
`numRows` and `numCols`	return integers, like `nrow` and `ncol`, except that they also work on atomic vectors (`numRows` returns the length of the vector, and `numCols` returns 1).
`rowIds` and `colIds`	return the IDs of the rows and columns. These are often character vectors, but need not be, depending on the class of `x`. They are like the components of `dimnames`, except that for named vectors, `rowIds` returns or sets the names and `colIds` returns `NULL`.
`colnames` and `rownames`	return the same values as `colIds` and `rowIds`, respectively.

See Also

`as.data.frame`, `matrix`, `Subscript`, `nrow`, `dimnames`.

Examples

```
x <- 1:10
y <- list(a=1:10, b=11:20)
is.rectangular(x)
y <- as.rectangular(y)
subscript2d(x,3,1)
subscript2d(y,4,1) <- 55
numRows(x)
numCols(y) <- 3
rowIds(x) <- letters[1:10]
colIds(y)
z <- cbind(1,1:4)
colnames(z)
colnames(z) <- colnames(z)
rownames(z) <- rownames(z)
```

`djia` *Dow Jones Industrial Average*

Description

High, low, opening, and closing prices and trading volume for the Dow Jones Industrial Average.

The data set has:

- The closing price only from 1915 through September 1928.
- The high, low, and closing prices from October 1928 through March 9, 1984.
- The high, low, opening, and closing prices from March 12, 1984 through December 1986.
- The high, low, opening, and closing prices and the trading volume from January 1987 through February 1990.

Format

An object of class `timeSeries` with the high, low, open and close prices stored as a `data.frame`.

Source

From Ohio State University web site, http://www.cob.ohio-state.edu/~fin/osudown.htm, downloaded in early 1990.

`exch.rate` *Foreign Exchange Rates*

Description

Exchange rates between the US Dollar and the

- British Pound (GBP)
- Canadian Dollar (CAD)
- German Mark (DEM)
- Japanese Yen (JPY)
- Swiss Franc (CHF)

in a multi-variate time series.

Data from Andreas S. Weigend, Bernardo A. Huberman, and David E. Rumelhart, "Predicting Sunspots and Exchange Rates with Connectionist Networks", pp. 395-432 in M. Casdagli and S. Eubank, eds, Nonlinear Modeling and Forecasting, Addison-Wesley, 1992.

`fed.rate` *Federal Reserve Interest Rates*

Description

Interest rate data from the web site of the Federal Reserve Bank, `https://www.federalreserve.gov/releases/h15/data.htm`, running from 1972 to 1997.

You can find more information on that web site.

Documentation below is derived from the data files.

`fed.rate` is a multi-variate time series, with the following columns:

Value

`prime.rate`	Bank Prime Loan Rate, daily including weekends and holidays.
`discount.rate`	Discount Rate for the Federal Reserve Bank of New York, daily including weekends and holidays, which is the simple interest rate at which depository institutions borrow from the Federal Reserve Bank of New York.
`fedfunds.rate`	Federal Funds Effective Rate, daily including weekends and holidays, which is the cost of borrowing immediately available funds, primarily for one day. The effective rate is a weighted average of the reported rates at which different amounts of the day's trading through New York brokers occurs.
`mortgage.rate`	Conventional Mortgage Rates for fixed-rate mortgages from the Federal Home Loan Mortgage Corporation, weekly on Fridays.

`hloc` *High, Low, Open, and Close Calculation*

Description

Calculates the high, low, first, and last elements of a vector. Especially useful for financial trading data in conjunction with the `aggregateSeries` function.

Usage

```
hloc(x)
```

Arguments

`x`	a vector for which to calculate high, low, open, and close.

Value

returns a vector with four elements:

the maximum value in `x`.

the minimum value in `x`.

the first value of `x`.

the last value of `x`.

`x` can be an array, but dimensions are ignored.

See Also

`aggregateSeries`.

Examples

```
x <- c(5, 2, 3, 6, 3, 2, 1, 7, 1)
hloc(x)
```

`net.packet` *Network Packet Traffic*

Description

Time, type, and size for 10,000 network packets on a local intranet, just before 5PM on March 27, 1998, as reported by the Unix snoop command. Size is available only for packets whose type is `"TCP"`.

Format

An object of class `timeSeries` with the high, low, open and close prices stored as a `data.frame`.

`positions` *Positions of* `series` *Objects*

Description

Accesses the positions of `series` objects.

Usage

```
positions(object)
positions(object) <- value
```

Arguments

`object`	the object for which to find positions.
`value`	the value to which to set the positions.

Details

This function can also be used on the left side of an assignment to set the positions of a `series` object.

Value

returns the `positions` slot of `object`.

See Also

`seriesData`, `timeSeries`, `signalSeries`.

Examples

```
x <- signalSeries(pos=1:10, data=11:20)
positions(x)
positions(x) <- 11:20
```

`say.wavelet` *Speech Signal*

Description

A signal of a voice saying the word `"wavelet"` , sampled at 11025 Hz. There are 8192 samples, ranging in time from 0 to approximately 0.7429 seconds.

Format

The signal is of class `signalSeries` .

`series-class` *Base Class for Time Series and Signals*

Description

A base class representing ordered data objects, such as time series and signals, that have positions (x values, times), and for each position a set of variables (stored in any rectangular data object).

Details

The `series` class holds x positions and variable data. It is valid only when the lengths of the positions and data match, and when the data slot is a rectangular object.

`seriesVirtual` is a virtual class corresponding to `series`. All of the methods for `series` objects are defined on the corresponding virtual `seriesVirtual` class so they can be inherited easily by extending classes.

`series` has two built-in extending classes: `timeSeries` and `signalSeries`. `series` is not meant to be used directly. Instead, most users should use the `signalSeries` and `timeSeries` classes. Extending classes should include both `series` and `seriesVirtual` in their representations.

Slots

data (`ANY`) the variable data, which can be any data object for which `is.rectangular` is `TRUE`, such as a `data.frame`, `matrix`, or atomic vector.

positions (`positions`) the x values for the variables.

start.position (`positions`) the starting x value.

end.position (`positions`) the ending x value.

future.positions (`positions`) future x values used for predictions.

units (`character`) units for the data.

title (`character`) title of the data set.

documentation (`character`) user-supplied documentation.

attributes (`ANY`) attributes slot for arbitrary use.

Series functions

- The `series` class has a validity function, `seriesValid`.
- The access functions `positions` and `seriesData` can access the positions and data in the object, and they can be used on the left side of assignments.
- There are also methods defined for `series` objects for the following functions:
  - `nrow`
  - `ncol`
  - `start`
  - `end`
  - subscripting
  - the standard rectangular data functions (see `is.rectangular`)
  - basic arithmetic.

See Also

`timeSeries` class, `signalSeries` class, `is.rectangular`.

`seriesData` *SeriesData of* `series` *Objects*

Description

Accesses the seriesData of `series` objects.

Usage

```
seriesData(object)
asSeriesData(object)
seriesData(object) <- value
seriesDataNew()
seriesDataValid(object)
```

Arguments

`object`	the object for which to find `seriesData`.
`value`	the value to which to set `seriesData`.

Details

This function can also be used on the left side of an assignment to set the `seriesData` of a `series` object.

Value

returns the `seriesData` slot of `object`.

See Also

`positions`, `timeSeries`, `signalSeries`.

Examples

```
x <- signalSeries(pos=1:10, data=11:20)
seriesData(x)
seriesData(x) <- 1:10
```

`seriesLag` *Time Series Lag/Lead Function*

Description

Returns a lagged/leading `timeSeries` or `signalSeries` object.

Usage

```
seriesLag(X, k = 1, trim = FALSE, pad = NA)
```

Arguments

`X`	an object of class `timeSeries` or `signalSeries`.
`k`	the number of positions the new time series or signal series is to lag or lead the input series, with a positive value resulting in a lagged series and a negative value resulting in a leading series.
`trim`	a logical flag. If `TRUE`, the missing values at the beginning or end of the returned series will be trimmed. The default is `FALSE`.
`pad`	any padding to fill in the beginning or ending missing values. The default is `NA`.

Details

The difference between `shift` and `seriesLag` is that the returned series of `shift` is shifted in time (`position`) while the returned series of `seriesLag` shifts the entire data slot but keeps the same time (`position`) intact. They all work for both `timeSeries` and `signalSeries` objects.

Value

returns a lagged or leading time (signal) series of the original data.

See Also

`lag`, `shift` .

Examples

```
x <- timeSeries(data=data.frame(x=1:10, y=11:20),
    from="7/4/2000", by="bizdays")
seriesLag(x, 1)
seriesLag(x, -1)
```

`seriesLength` *Length of a* `timeSeries`

Description

Returns the length of a `timeSeries`; that is, it returns the number of positions in the `timeSeries`.

Usage

```
seriesLength(x)
```

Arguments

`x`	an object of class `timeSeries`.

Value

returns the length of the `timeSeries`.

Note

This function is distinquished from the `length` function, which returns the number of series in the `timeSeries` object.

See Also

`timeSeries` class.

Examples

```
x <- timeSeries(data=data.frame(x=1:10, y=11:20), from="7/4/2000", by="bizdays")
seriesLength(x)
length(x)
```

`seriesMerge` *Merging for Time Series and Signals*

Description

Merges time series or signal objects, making a new object with all the columns of the input objects, and some or all of the rows, depending on how their positions match.

Usage

```
seriesMerge(x1, x2, ..., pos=positions(x1), how,
        error.how, localzone=FALSE, matchtol=0,
        suffixes)
```

Arguments

`x1`	the first object to be merged.
`x2`	the second object to be merged.
`...`	the other objects to be merged.
`pos`	the positions to align to, or `"union"` to make a union of all input positions. (The default argument values give an intersection of all the positions.)
`how`	after the positions to align to are determined, `how` determines how to treat positions that are missing from the various input objects. Can be one of the following:

`"NA"`	Inserts a row of `NA`.
`"drop"`	Drops that position entirely.
`"nearest"`	Uses the row with the nearest position.
`"before"`	Uses the data from the row whose position is just before the unmatched position.
`"after"`	Uses the data from the row whose position is just after the unmatched position.
`"interp"`	Interpolates linearly between `"before"` and `"after"`.

The default is `"drop"` unless `pos="union"`, in which case `"drop"` makes no sense and the default is `"NA"`.

`error.how`	specifies what to do in the event of an out of bounds error, which can occur when `how` is `"before"`, `"after"`, or `"interp"`. Can be one of the following:

`"NA"`	Inserts a row of `NA`
`"drop"`	Drops that position entirely
`"nearest"`	Uses the row with the nearest position.

The default is `"drop"` unless `pos="union"`, in which case `"drop"` makes no sense and the default is `"NA"`.

`localzone`	if `TRUE` (that is, all input positions must be calendar-based), merge by matching/interpolating with all positions in their local time zones, instead of with the absolute GMT times.
`matchtol`	the tolerance for matching positions. Positions that match within `matchtol` do not invoke `how` argument methods.
`suffixes`	the suffixes to append to the column names that are duplicated between the various input data objects. The default value is `paste(".",1:nargs,sep = "")`, where `nargs` is the total number of data arguments.

Value

returns a new `series` object containing all the columns of all the inputs, and all the rows of all the inputs, according to the alignment methods described above.

See Also

timeSeries, signalSeries, positions, align, merge.

Examples

```
a <- signalSeries(pos=1:10, data=data.frame(a = 11:20, b = 5 * (1:10)))
b <- signalSeries(pos=5:14, data=data.frame(a = 11:20, b = 5 * (1:10)))
seriesMerge(a, b)
a <- timeSeries(pos=as(1:10, "timeDate"),
data=data.frame(a = 11:20, b = 5 * (1:10)))
b <- timeSeries(pos=as(5:14, "timeDate"),
        data=data.frame(a = 11:20, b = 5 * (1:10)))
seriesMerge(a, b, pos="union")
```

`shift` *Create a Shifted Time Series*

Description

Returns a time series like the input but shifted in time.

Usage

```
shift(x, k=1)
```

Arguments

`x`	a univariate or multivariate regular time series. Missing values (`NA`s) are allowed.
`k`	the number of positions the input series is to lead the new series. That is, the resulting series is shifted forwards in time; negative values lag the series backwards in time. Non-integer values of `k` are rounded to the nearest integer.

Details

`shift` is a generic function. Its default method calls `lag(x,-k)`.

`shift` also has a method for `series` objects, which works for both `timeSeries` and `signalSeries` objects.

- To align the times of several new-style time series, use `seriesMerge`.
- To align the times of several old-style time series, use `ts.intersect` or `ts.union`.
- To compute a lagged/leading series with same time position but shifted data slot, use `seriesLag`. (`seriesLag` also works for both `timeSeries` and `signalSeries` objects.)

Value

returns a time series with the same data as `x`, but with positions lagged by `k` steps.

Note

The `shift` function replaces the `lag` function, which illogically had the opposite sign of shifting. (The `lag` function has been retained only because it is used in other functions.)

See Also

`seriesMerge`, `lag`, `lag.plot`, `ts.intersect`, `ts.union`.

Examples

```
x <- signalSeries(data=data.frame(a=1:10, b=letters[1:10]), positions=1:10)
x5 <- shift(x,5)
seriesMerge(x, x5, pos="union")
```

`signalSeries` *Create a* `signalSeries` *object*

Description

Creates an object of class `signalSeries`

Usage

```
signalSeries(data, positions., units, units.position, from = 1, by = 1)
```

Arguments

`data`	(`ANY`) the variable data, which can be any data object for which `is.rectangular` is `TRUE`, such as a `data.frame`, `matrix`, or atomic vector.
`positions.`	(`positions`) the x values for the variables, which must be of type `positionsNumeric`. If not given, then the `positions` are computed using the `numSeq` function with the `from` and `by`.
`units`	(`character`) the units for the data.
`units.position`	(`character`) the units for the `positions` slot.
`from`	the start of the sequence.
`by`	the increment for the sequence.

See Also

`signalSeries` class, `numericSequence` class.

Examples

```
signalSeries( pos=1:10 , data = 1:10)
signalSeries(data=data.frame(x=1:10, y=11:20), from=2, by=2)
```

`signalSeries-class` *signalSeries Class*

Description

Represents non-calendar time series and signal objects.

Details

The `signalSeries` class inherits from the `series` and `seriesVirtual`classes. It has slots that hold x positions and variable data inherited from the `series` class.

A `signalSeries` object is valid only when the lengths of the positions and data match, the data is a rectangular object, and the `positions` slot holds a `positionsNumeric` object.

Slots

All slots except the last, `units.position`, come from the `series` object.

data (`ANY`) the variable data, which can be any data object for which `is.rectangular` is `TRUE`, such as a `data.frame`, `matrix`, or atomic vector.

positions (`positions`) the x values for the variables, which must be of type `positionsNumeric`.

start.position (`positions`) the starting x value.

end.position (`positions`) the ending x value.

future.positions (`positions`) future x values used for predictions.

units (`character`) the units for the data.

title (`character`) the title of the data set.

documentation (`character`) user-supplied documentation.

attributes (`ANY`) the attributes slot for arbitrary use.

units.position (`character`) the units for the `positions` slot.

signalSeries functions

You can create objects of class `signalSeries` using the `new` function, in which case they are set up to be empty. Alternatively, you can create objects of class `signalSeries` using the `signalSeries` function.

These objects can be subscripted and used in mathematical operations much like data frames or matrices.

See Also

`series` class, `signalSeries`, `is.rectangular`.

`tbauc.3m` *Treasury Bill Auction Rates*

Description

Treasury Bill auction rate data running from 1980 to 1997 for 3 month, 6 month, and 1 year durations.

Format

Three separate `timeSeries` objects with Treasury Bill auction rates:

tbauc.3m Average of interest rate bids accepted in regular treasury auctions of 13-week bills (also known as 3-month bills). Currently, the auctions are held each Monday for bills to be issued the ensuing Thursday, in the absence of holidays.

tbauc.6m Average of interest rate bids accepted in regular treasury auctions of 26-week bills (also known as 6-month bills). Currently, the auctions are held each Monday for bills to be issued the ensuing Thursday, in the absence of holidays.

tbauc.1y Average of interest rate bids accepted in regular treasury auctions of 52-week bills (also known as 1-year bills). Currently, the auctions are held at roughly monthly intervals.

Source

From the web site of the Federal Reserve Bank, `https://www.federalreserve.gov/releases/h15/data.htm`.

`tbond` *Treasury Bond Futures Trading Data*

Description

Treasury Bond futures trading data: high and low prices over 20-minute intervals from January 7, 1994, to February 3, 1995.

Taken from the Ohio State University web site http://www.cob.ohio-state.edu/~fin/osudown.htm (downloaded downloaded in early 1996). The dataset is a two-column time series. It is used to illustrate a drop in bond prices that occurred in 1994.

`tcm.curve` *Treasury Constant Maturity Curve*

Description

Treasury Constant Maturity Curve data running from 1982 to 1997. The Constant Maturity Curve data come from yield curves constructed by the U.S. Treasury Department from the yields of actively traded issues adjusted to constant maturities.

Format

`tcm.curve` is a multivariate `timeSeries` object with the following columns:

three.month Three-month rate.

six.month Six-month rate.

one.year One-year rate.

two.year Two-year rate.

three.year Three-year rate.

five.year Five-year rate.

seven.year Seven-year rate.

ten.year Ten-year rate.

twenty.year Constructed from the 20-year Treasury department numbers, based on the 20-year bond through December 1986 (at which time the 20-year bond was discontinued), and from the new computation starting in October of 1993 based on outstanding bonds with approximately 20 years remaining to maturity. There is no data between 1987 and September 1992.

thirty.year Thirty-year rate.

long.term An unweighted average of rates on all outstanding bonds neither due nor callable in less than 10 years, also calculated by the Treasury Department.

Source

From the web site of the Federal Reserve Bank, `https://www.federalreserve.gov/releases/h15/data.htm`.

timeSeries *Create a* timeSeries *Object*

Description

Creates an object of class timeSeries.

Usage

```
timeSeries(data, positions., units., from = timeCalendar(d = 1,
    m = 1, y = 1960), by = "days", k.by = 1, align.by = FALSE,
    week.align = NULL)
```

Arguments

data	(ANY) the variable data. Can be any data object for which is.rectangular is TRUE, such as a data.frame, matrix, or atomic vector.
positions.	(positions) the x values for the variables. Must be of type positionsCalendar. If not given, then the positions are computed using the timeSeq function with the from, by, k.by, align and week.align arguments.
units.	(character) the units for the data.
from	the starting value of the sequence. A timeDate object (or number or character string representing one).
by	the spacing between successive values in the sequence. Can be a timeSpan, timeRelative, or numeric value, in which case k.by is ignored. Alternatively, it can be one of the following character strings: • "milliseconds" • "seconds" • "minutes" • "hours" • "days" • "weekdays" • "bizdays" • "weeks" • "months" • "quarters" • "years" giving the time units of intervals between values in the sequence.
k.by	a non-zero integer giving the width of the interval between consecutive values in the sequence in terms of the units given in by. Ignored if by is not a character string.

`align.by`	a logical value. If `TRUE`, adjusts the sequence so that each element is on a whole number of the `by * k.by` units. For example, if the units are 2 months, the sequence is only on the first of January, March, and so on. Ignored if `by` is not a character string.
`week.align`	if `by` is `"weeks"`, you can supply a character string (or a number, 0 to 6 with 0 being Sunday) to specify a weekday to use. (The character string must be sufficient to make a unique case-insensitive match to the strings in `timeDateOptions("time.day.name")`.) • If `align.by` is `FALSE`, the sequence is adjusted so that all its elements fall on the given weekday. • If `align.by` is `TRUE`, the sequence is adjusted to start at midnight. In either case, the `extend` argument is used to decide which direction to adjust the day. This argument is ignored if `by` is not a character string, or if it is not `"weeks"`.

See Also

`timeSeries` class, `timeSequence`.

Examples

```
timeSeries( pos=timeCalendar( d=1:10 ), data = 1:10)
timeSeries(data=data.frame(x=1:10, y=11:20), from="7/4/2000", by="bizdays")
```

`timeSeries-class` *Calendar Time Series Class*

Description

Represents calendar time series objects.

Details

The `timeSeries` class inherits `series` and `seriesVirtual`. From `series`, it inherits slots that hold x positions and variable data. A `timeSeries` object is valid only when the lengths of the positions and data match, the `data` slot is rectangular, and the `positions` slot holds a `positionsCalendar` object.

Slots

All slots except the last two, `fiscal.year.start` and `type`, are inherited from the base `series` class.

data (`ANY`) the variable data, which can be any data object for which `is.rectangular` is `TRUE`, such as a `data.frame`, `matrix`, or atomic vector.

positions (`positions`) the x values for the variables, which must be of type `positionsCalendar`.

start.position (`positions`) the starting x value.

end.position (`positions`) the ending x value.

future.positions (`positions`) future x values used for predictions.

units (`character`) the units for the data.

title (`character`) the title of the data set.

documentation (`character`) user-supplied documentation.

attributes (`ANY`) the attributes slot for arbitrary use.

fiscal.year.start (`numeric`) the month number for fiscal year start.

type (`character`) the type of time series.

Series functions

You can create objects of class `timeSeries` using the `new` function, in which case they are set up to be empty and have their fiscal year starting in January. Alternatively, you can create objects of class `timeSeries` using the `timeSeries` function.

These can be subscripted and used in mathematical operations much like data frames or matrices.

See Also

`series` class, `timeSeries`, `is.rectangular`.

`ts.update` *Update Old* `ts` *Objects*

Description

Converts an old `ts` object to a `signalSeries` object.

Usage

```
ts.update(x)
```

Arguments

`x`	the time series to convert.

Value

returns a `signalSeries` object with equivalent positions.

See Also

`signalSeries`.

Examples

```
ts.update(ts(1:10))
```

`unionPositions` *Positions Object Union With Tolerance*

Description

Makes a union of numeric or calendar positions (that is, positions of series objects (which can be numeric), time vectors, or sequences) objects using `localzone` and `matchtol` as in the `seriesMerge` and `align` functions.

Usage

```
unionPositions(..., localzone = FALSE, matchtol = 0)
```

Arguments

`...`	the positions objects to be joined.
`localzone`	a logical value. If `TRUE`, creates a union by matching with all passed-in positions in their local time zones, instead of with the absolute GMT times. (The positions must be calendar-based.)
`matchtol`	the tolerance for matching positions. Positions that match within `matchtol` are not duplicated in the output.

Value

returns a new positions object containing all of the input positions, with duplicates (as defined by `matchtol` and `localzone`) removed.

returns `numeric(0)` if no `...` arguments are given.

See Also

`positions` class, `align`, `seriesMerge`.

Examples

```
unionPositions(1:10, 5:20)
unionPositions(1:10, 5.1:20.1, matchtol=.3)
unionPositions(timeCalendar(d=1:10), timeCalendar(d=5:20))
unionPositions(timeCalendar(d=1:10, zone="PST"),
               timeCalendar(d=5:20, zone="EST"))
unionPositions(timeCalendar(d=1:10, zone="PST"),
               timeCalendar(d=5:20, zone="EST"), localzone=TRUE)
```

Index